Improvements in Targeting Precision by Applying Pointing Corrections to a Computer Assisted Rehabilitation Environment (CAREN)

Delwin Johnson, MS
Pinata Sessoms, PhD
LT Seth Reini, PhD

Naval Health Research Center

Report No. 13-58 supported by Defense Health Program (6.7 Funding) under Work Unit No. 61030.

Naval Health Research Center
140 Sylvester Road
San Diego, CA 92106-3521

Improvements in Targeting Precision by Applying Pointing Corrections to a Computer Assisted Rehabilitation Environment (CAREN)

Delwin Johnson, MS, Pinata Sessoms, PhD, and LT Seth Reini, PhD
Naval Health Research Center

Summary

The Computer Assisted Rehabilitation Environment (CAREN) at the Naval Health Research Center (NHRC) is an immersive virtual environment that allows for physical and cognitive rehabilitation as well as research. One of the capabilities of the CAREN system is a 3D pointing module that can be used for pointing or shooting at targets within the CAREN 3D virtual environment. However, the overall accuracy of the CAREN targeting is not as precise as would be desired for a simulated shooting system. This paper describes a method to improve the accuracy of CAREN targeting by first measuring the targeting errors and then correcting the errors through the use of a scripting module in the CAREN software. The initial implementation of pointing corrections produced favorable results overall for the CAREN at NHRC.

Background and Purpose

The CAREN at NHRC is an interactive virtual environment that has the ability to task subjects both cognitively and physically. The CAREN D-Flow software (Motek Medical BV, Amsterdam, The Netherlands), which allows the operator to create and control the scenarios, has various programming modules from which to manipulate and monitor the hardware components, activate events, or record information. The D-Flow pointing module (Pointer3D) targets virtual objects in a CAREN scene by extrapolating the position of two markers attached to a simulated weapon or other pointing device as shown in Figure 1. Distortions in the CAREN targeting system typically result in the projected target not aligning well with the pointing markers, thus limiting the practical use of the CAREN in studies requiring precision targeting. The purpose of this paper is to describe a procedure that can be used to improve accuracy of the existing targeting system in a manner that is portable between D-Flow applications. This will enable more effective use of the CAREN in protocols requiring accurate reproduction of shooting dynamics.

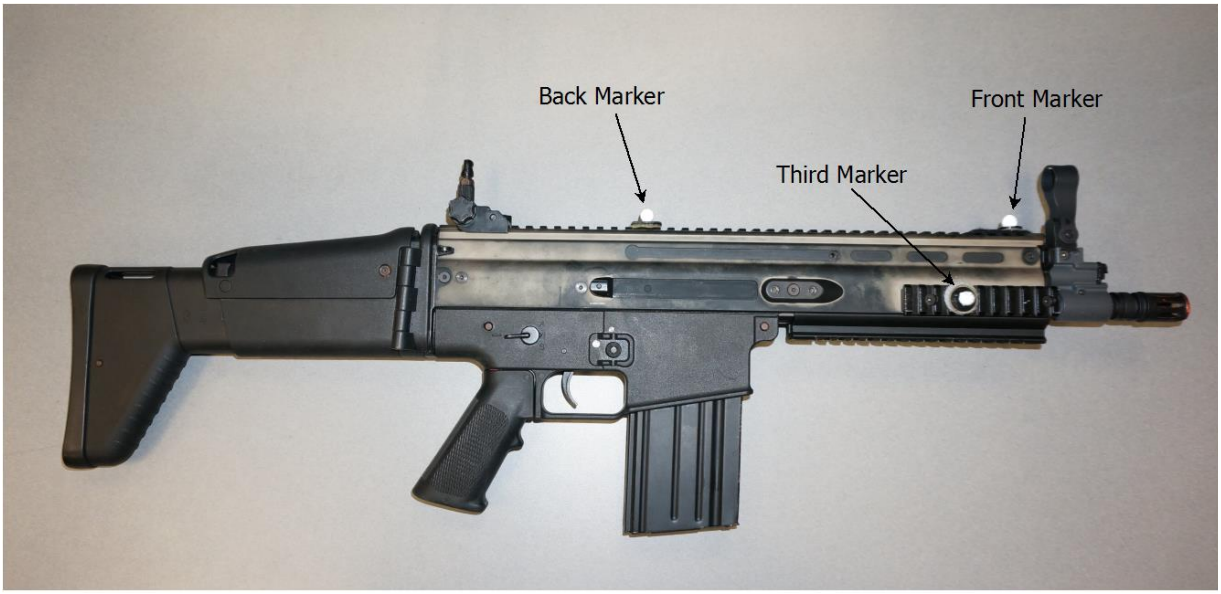

Figure 1. A simulated weapon with front and back pointing markers. The back marker is also used to determine the subject's fore-aft position on the treadmill for purposes of pointing corrections. The optional third marker is used to identify the front and back pointing markers in the motion capture system.

Approach

The D-Flow Pointer3D module pointing errors may be caused by the position of the markers on the pointing device, the motion capture system calibration, the D-Flow scene environment, the curvature of the screen, or the geometric distortion of the video projectors. Corrective action may be based on the translational position of the pointing device, the angular position of the pointing device, or the projected intersection of the targeting ray with the screen. Empirical observations suggest that the targeting errors are primarily based on the screen intercept position. It was also observed that the targeting errors varied with the translational location of the pointing device, in particular the fore-aft position of the subject along the treadmill. The pointing errors appear to be primarily an artifact of the CAREN virtual 3D environment, but they can also be caused by marker misalignment and poor calibration of the motion capture system.

The approach to improve the targeting is to measure pointing errors along an array of screen intercept positions based on a defined grid, and then use a D-Flow Lua scripting module to apply a virtual positional correction to the front pointing marker so that the displayed crosshairs on the screen align with the weapon sight for all areas of the projection screen. The pointing correction is based on the screen intercept position and is composed of a summation of an overall null offset

(vertically and horizontally) plus arrays of vertical and horizontal corrections per screen grid location. This summation simplifies recalibration for the cases of switching or adjusting pointing devices as one would only have to update the two null offset values rather than having to reconstruct the entire correction array. The pointing correction grids are measured with the pointing device at a forward position and also at an aft position on the platform to compensate for targeting error dependency on fore-aft treadmill position.

Having measured actual required pointing corrections at a collection of grid points over the CAREN projection screen, a method of correcting any position on the screen must be implemented. One method is to fit a 2D polynomial equation to the measured points. However, a preferred method, as used on the CAREN at NHRC, is to employ a bilinear interpolation between the measured screen grid positions. This solution will always pass through the actual measured points and is simple to maintain should it become necessary to modify the pointing correction arrays. Though it is easier to implement this method against a rectangular grid, the grid does not have to be evenly spaced so there remains some ability to concentrate measured points in areas of high interest.

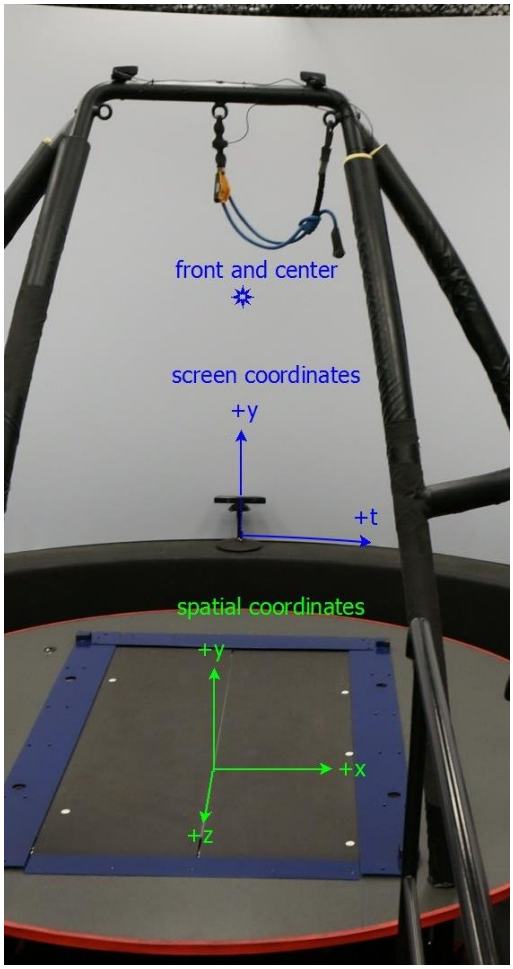

Figure 2. CAREN spatial (green) and cylindrical screen (blue) coordinates. The null offset values are measured at the front and center position on the screen.

Geometric Overview

Figure 2 illustrates the two coordinate systems referenced in this paper. The spatial coordinates are described by Cartesian axes *x*, *y*, and *z*, with the origin located at the center of the CAREN platform. The vertical axis is designated as *y* so as to remain consistent with the CAREN coordinate system. The CAREN projection screen at NHRC is cylindrical with radius *r*, and so it is natural to describe the screen intercept position in cylindrical coordinates *t* (theta) and *y*. The axis of the screen cylinder passes through the Cartesian origin and the vertical Cartesian *y* values are equivalent to the cylindrical *y* values. The horizontal position on the screen may be described either as a Cartesian position *x*, *z*, or more simply as the cylindrical angle *t*. Theta is zero at the center of the screen ($x = 0$, $z = -r$) and positive to the right as viewed from the origin. The vertical position on the screen is described as positive values of *y*, with $y = 0$ at the

bottom of the screen. The CAREN at NHRC has a screen radius of 2.5 m, a viewing angle of $-90^o \geq t \geq 90^o$, and a screen image height ranging from $0 \geq y \geq 2.5$ m. The front and center screen null offset calibration position ($t = 0^o$, $y = 1.25$ m at NHRC) is also indicated in Figure 3.

The pointing device, typically a mock weapon, defines a pointing ray that will generally intercept the projection screen. An overhead view of the screen cylinder allows for the *x*, *z* (or *t*) projection screen intercept to be calculated as a ray intercepting a 2D circle from the inside. Simultaneous equations describing two straight lines and a circle (Eqs 1–3) form the quadratic equation shown as Eq (4).

$$p_x = b_x + mv_x \quad (1)$$

$$p_z = b_z + mv_z \quad (2)$$

$$(x - x_c)^2 + (z - z_c)^2 = r^2 \quad (3)$$

$$m^2(v_x^2 + v_z^2) + 2m(v_x b_x + v_z b_z) + b_x^2 + b_z^2 - r^2 = 0 \qquad \text{for } x_c = 0, z_c = 0 \quad (4)$$

where:

p = projected pointing ray,
b = back marker,
v = pointing vector (front marker – back marker),
m = multiplier required to intercept screen,
x_c, z_c = coordinates of the circle center, and
r = screen radius.

For the CAREN at NHRC, $x_c = 0$, $z_c = 0$ and $r = 2.5$ m. A solution for multiplier *m* is sought for the quadratic equation where the quadratic discriminant is positive (indicating an interception) and *m* is positive (indicating forward projection). Screen intercepts s_x, s_y, and s_z are then calculated as described in Eqs (5–7).

$$s_x = b_x + mv_x \quad (5)$$

$$s_y = b_y + mv_y \quad (6)$$

$$s_z = b_z + mv_z \quad (7)$$

Further restrictions require that pointing be toward the screen ($-90^o \geq \theta \geq 90^o$) and a nonzero pointing angle from vertical is enforced. This method may be modified for spherical projection screens as detailed in Appendix A.

Measurement of Targeting Errors

A D-Flow software application was written for the purpose of measuring and correcting targeting errors. The user typically positions a mock weapon to achieve the desired nominal screen intercept position and back marker *z* value (i.e., subject's fore-aft position), and then adjusts the displayed target with a game controller joystick until the displayed target aligns with the weapon sights. Another button press on the game controller is used to write the pointing correction data to a file. The user should strive to maintain a normal shooting position with the weapon stock located near the centerline of the treadmill ($x \approx 0$). The D-Flow targeting application displays the nominal *t* and *y* screen position, the *dt* and *dy* values required to correct the target, and also the *z* value of the back marker *(*denoted as *bmz* in the D-Flow application and Lua script) as shown in white print in Figure 3. For example, the screen intercept position in Figure 3 is $t = 41^{o}$ to the right of center and $y = 1.64$ m above the base, and the back marker on the weapon is located at bmz or $z = -0.24$ m (forward from the origin). A pointing correction of $dt = 0.0^{o}$ and $dy = -0.02$ m has been applied to match the mechanical aiming sights of the weapon to the projected crosshairs on the screen for this particular area of the screen shown in Figure 3.

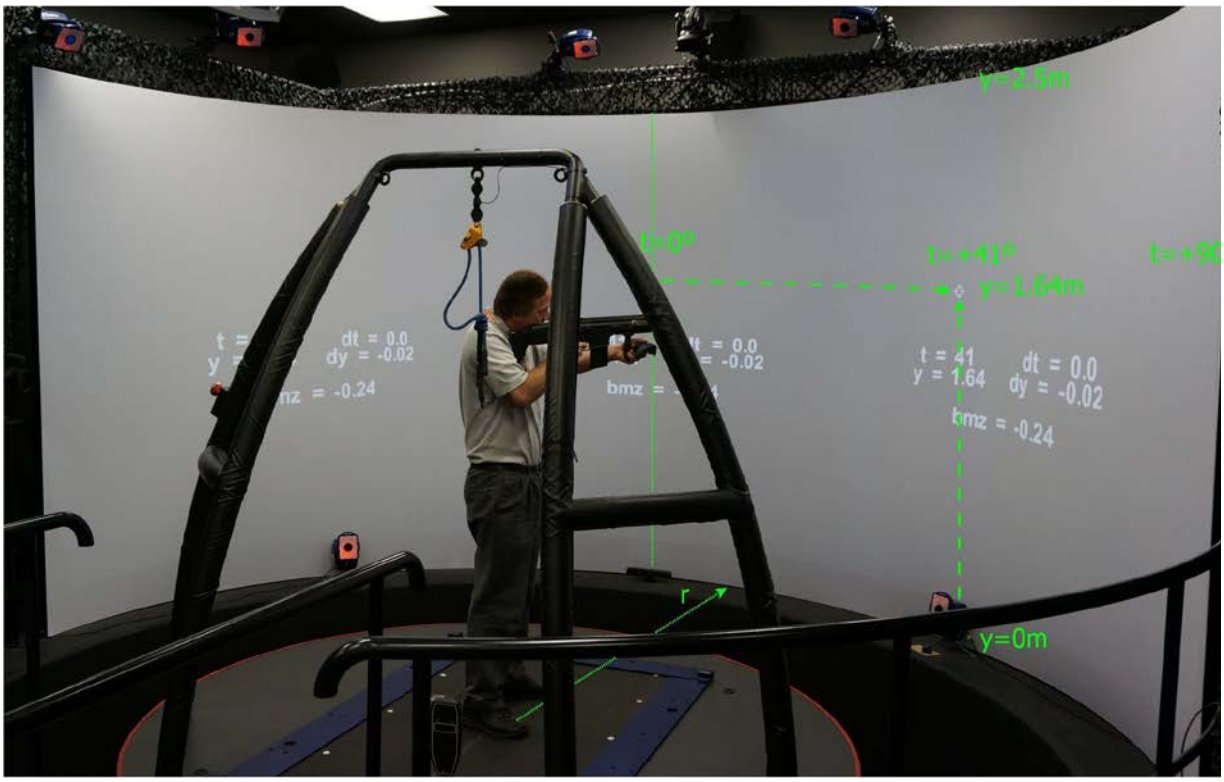

Figure 3. Measuring the CAREN targeting error in the D-Flow pointing correction application. The real-time data from the CAREN targeting error measurement application is displayed in white.

Targeting error is first measured at the front and center screen position with all pointing corrections disabled. The *dt* and *dy* values measured at the front and center position are the null offset values which offer a simple method of compensating for variations in marker placement on the pointing weapon. The example null offset values of *dt_null* = 0.8º and *dy_null* = -0.13 m, shown in Table B1 of Appendix B and also in lines 19 and 20 of the Lua code listed in Appendix C, were collected in the forward position on the treadmill. The actual position of the measured null values is arbitrary, but should be consistent when measuring new null values. The assignments in lines 19 and 20 of the Lua code must be updated with the new *dt_null* and *dy_null* values before continuing. If a different weapon is used for targeting, or if markers on a given weapon are disturbed, then it should only be necessary to update the two front and center null offset values rather than collecting new values for the entire arrays. Alternatively, the user may simply adjust the mechanical sights on the weapon.

Two fore-aft treadmill locations were used for the pointing correction arrays on the NHRC CAREN and were defined as z = -0.40 m and z = +0.10 m, respectively, as indicated above the arrays listed in Appendix C. The subject position is determined by the *z* coordinate value of the back marker on the pointing weapon, and is designated by the abbreviation *bmz*. These two locations were chosen for the correction arrays as they approximated the fore-aft range on the treadmill that the subjects normally walked while shooting.

The full screen survey is then performed, which involves measuring the targeting errors for all desired grid locations on the screen for both fore-aft subject positions on the treadmill. The nominal screen position, treadmill position, and measured targeting error values are recorded to a file using a game controller action button recognized by D-flow. The D-Flow application displays and records the *dt* and *dy* as the measured targeting error value minus the previously measured overall *dt_null* and *dy_null* offset values. The arrays shown in lines 47–75 of the Lua code are then manually updated with the new measured correction values. The targeting error grid measurement process is somewhat laborious, but should not have to be repeated in full unless the projectors or projection image are altered (e.g., blending and warping changes). Day-to-day calibration is normally limited to updating the pair of offset null calibration values.

The size of the measured targeting error arrays was selected to balance sufficient coverage density with time and labor constraints. The normal grids are composed of five rows and nine columns, which extend toward, but not entirely out to, the screen perimeter, with the intent of concentrating the measured grid points where the shooting is more likely to occur. The software will extend the outer measured grid correction value to the screen perimeter. The measured errors at the front and center positions are expected to be zero as these errors are accommodated by the null correction values. However, small nonzero error values at the front and center positions may be present due to random fluctuations.

Example measured targeting errors per screen position are shown in Appendix B. The null offset front and center values are shown in Table B1, and the measured targeting errors less the null offset values are shown in Tables B2 and B3. An error in θ of 1º is equivalent to about 4 cm on the screen. Residual targeting errors, shown in Tables B4 and B5, were measured after the new pointing corrections were applied. The residual errors were measured between the measured grid points and at a treadmill location between the two fore-aft treadmill positions so as to validate the interpolation techniques for both the screen locations and the subject location. Most of the residuals were very small, though it can be seen that there were significant residual horizontal errors at t = +30° along much of the screen height. This is a known projector blending problem zone at NHRC, which results in nonlinear horizontal targeting in this region of the screen. This problem was successfully addressed by inserting an additional grid column at θ = +30° as can be seen in the Lua code data arrays shown in Appendix C. Also note the inserted code near line 166 in the Lua code. The large horizontal residual value at θ = +50° and y = 0.5 m should also be addressed or at least identified as a screen area to avoid for targeting purposes.

The granularity of the targeting adjustments via the game controller joystick affects a target shift of about 0.1º or half a centimeter for the t and y axes on the NHRC CAREN projection screen. It takes about 10 clicks of the game controller to move the target from the center to the edge of the forward weapon sighting ring on a typical weapon used at NHRC. Targeting errors < 0.2º horizontal or 0.01 m vertical would be desirable, while targeting errors > 1.0º or 0.05 m indicate a significant targeting problem.

Implementation in Other D-Flow Applications

To implement the targeting corrections into another D-Flow application, such as the simple application shown in Figure 4, one simply inserts the Targeting Correction Lua scripting module into the desired D-Flow application and connects the outputs of this module (new marker positions) to the marker inputs of the Pointer3D module. The only required inputs to the scripting module are the front and back marker x, y, and z values shown in lines 2–7 of the Lua code in Appendix C. The correction on/off control input “cor” of line 10 should be unattached or set to zero in order to effect corrections. The effect of the correcting scripting module is to modify the front marker x, y, z values. The back marker x, y, z values are simply passed through the scripting module. The software developer has the option by assigning sliders in the user interface to allow for a convenient method of updating the null offset values (not implemented in the Lua code in Appendix C). The other scripting module inputs, used for measuring targeting errors and debugging purposes, are not required to be connected when implementing pointing corrections in D-Flow applications.

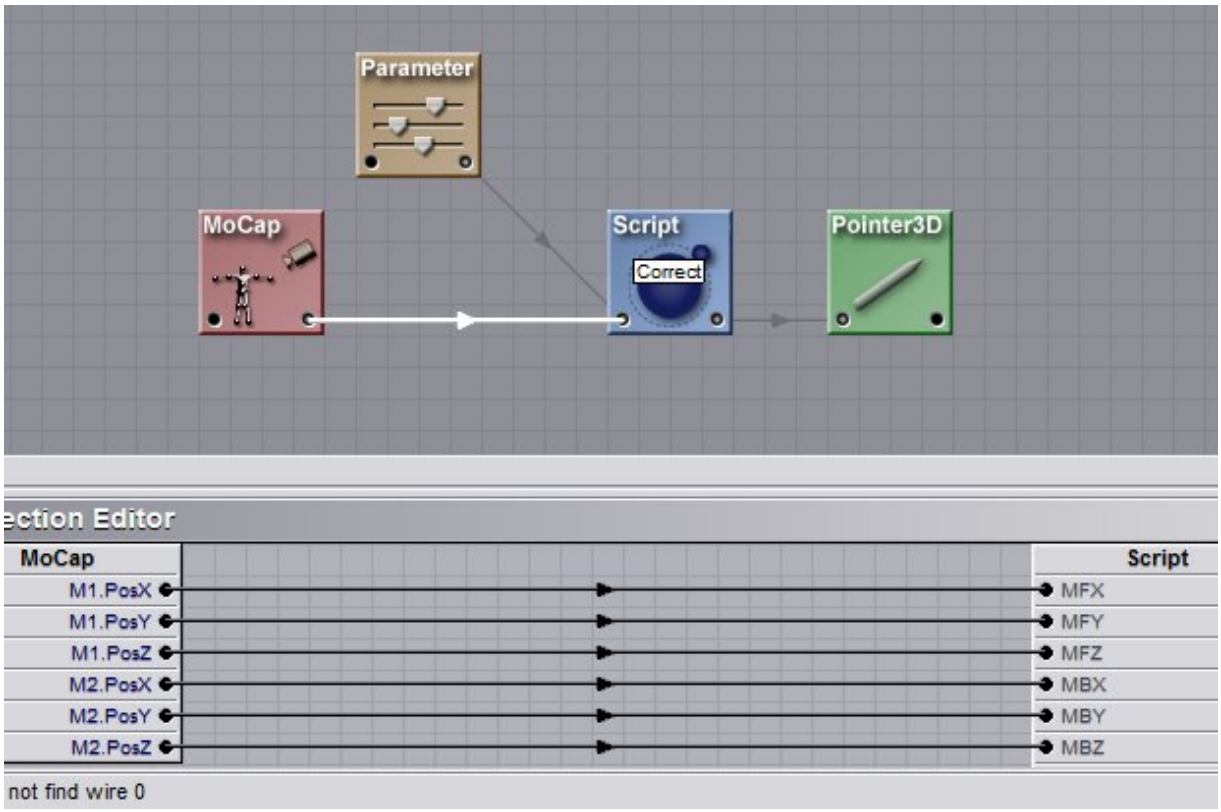

Figure 4. Implementing pointing corrections in a D-Flow software application. The positions of the front and back pointing markers are passed from the mocap module, into and out of the pointing correction Script module, and to the Pointer3D module. Optional graphical user interface sliders can be used to modify the null offset values in the Parameter module.

The required outputs are the front and back marker *x*, *y*, *z* values shown in lines 271–276 of the Lua code. The other outputs are for displaying screen intercept positions and targeting correction values when measuring targeting errors. These other outputs are not required to be connected when implementing pointing corrections in D-Flow applications.

Conclusions

Using the Pointer3D module within D-Flow applications is not sufficient to employ accurate shooting that aligns with the sight of a weapon. Corrections must be done to account for marker position on the weapon and screen warping. The measured targeting errors listed in Tables B2 and B3 in Appendix B indicate that simply subtracting out the front and center null error values does not produce satisfactory results, and thus it is necessary to apply a full array of targeting corrections per screen location. The effectiveness of applying the full targeting corrections was ascertained by measuring the residual targeting errors as listed in Tables B4 and

B5 of Appendix B. Error values listed in Table B4 indicate that the horizontal residual targeting errors on the left side of the screen were very small. However, there were some concerns on the right side of the screen. This result is consistent with known projector blending issues with the CAREN at NHRC. The vertical targeting error residuals listed in Table B5 were consistently small across the entire screen. The initial implementation of pointing corrections as described in this document produced favorable results overall for the CAREN at NHRC and offers credence that the CAREN could be used as an effective shooting simulator so that realistic targeting can be incorporated within CAREN software applications (Figure 5).

Figure 5. An application where the D-Flow pointing module is used to aim the weapon at targets. The projected target is indicated with a white circular symbol on the screen to the right of the subject.

Appendix A – Considerations for a Spherical Projection Screen

A pointing ray defined by a pair of pointing markers will intercept a spherical projection screen in a manner similar to that of the cylindrical screen described in the main text. The ray originates from the back pointer marker and points through the front marker as before. The front and back markers are arbitrarily located within the radius of the spherical screen, and the center of the spherical screen is defined as the origin. Simultaneous equations describing three straight lines (Eqs A1–A3) and a sphere (Eq A4) form the quadratic equation shown as Equation A5.

$$p_x = b_x + mv_x \quad \text{(A1)}$$

$$p_y = b_y + mv_y \quad \text{(A2)}$$

$$p_z = b_z + mv_z \quad \text{(A3)}$$

$$(x - x_c)^2 + (y - y_c)^2 + (z - z_c)^2 = r^2 \qquad x_c, y_c, z_c = 0 \quad \text{(A4)}$$

$$(v_x^2 + v_y^2 + v_z^2)m^2 + 2*(v_xb_x + v_yb_y + v_zb_z)\, m + b_x^2 + b_y^2 + b_z^2 - r^2 = 0 \quad \text{(A5)}$$

where:

p = projected pointing ray,
b = back marker,
v = pointing vector (front marker – back marker),
m = multiplier required to intercept screen,
x_c, y_c, z_c = coordinates of the sphere center, and
r = screen radius.

The original multiplier *m* value of unity simply defines the forward markers. The desired solution for multiplier *m* is sought to define the coordinates of the screen intercept. This is accomplished by solving the quadratic equation where the quadratic discriminant is positive (indicating an interception) and *m* is positive (indicating forward projection).

A spherical screen is naturally defined with spherical coordinates theta and phi (θ, ϕ), and thus the *x*, *y*, *z* screen intercept coordinates would need to be transformed to θ, ϕ. Using the CAREN conventions of the y-axis being vertical and the negative z-axis pointing toward the screen, the coordinate transformations are:

$$\phi = \arccos\left(y / (x^2 + y^2 + z^2)^{0.5}\right) \quad \text{(A6)}$$

$$\theta = \text{atan2}(x, -z) \quad \text{(A7)}$$

Likewise, the pointing correction grids would also be defined in terms of θ and ϕ. A nonzero pointing angle from vertical is required to avoid a coordinate singularity.

Appendix B – Examples of Measured Targeting Errors

Table B1. Null correction values for the front and center screen intercept position

dt null = 0.8º
dy null = -0.13 m

Table B2. Measured Horizontal Targeting Error Values per Screen Location (θ, degrees)

Height	-80º	-60º	-40º	-20º	0º	20º	40º	60º	80º
2.25 m	-0.8	-0.8	-0.8	-0.5	0.0	-0.1	-0.6	-0.5	-0.1
1.75 m	-0.4	0.0	-0.5	-0.5	0.0	-0.1	-0.6	-0.7	0.0
1.25 m	-0.4	0.1	-0.2	-0.4	0.1	0.1	-0.2	-0.6	0.0
0.75 m	-1.0	-0.1	-0.6	-0.5	0.1	0.3	0.0	0.1	0.7
0.25 m	-2.1	-1.1	-0.8	-0.9	-0.3	0.3	0.3	0.6	2.1

Table B3. Measured Vertical Targeting Error Values per Screen Location (*y*, meters)

Height	-80º	-60º	-40º	-20º	0º	20º	40º	60º	80º
2.25 m	-.04	-.05	-.05	-.05	-.04	-.03	-.04	-.04	-.03
1.75 m	-.01	-.04	-.02	-.02	-.01	.00	-.01	.00	-.01
1.25 m	.00	-.02	-.02	.00	.00	.01	.00	.00	-.01
0.75 m	-.03	-.04	-.04	-.05	-.05	-.04	-.05	-.01	-.05
0.25 m	-.07	-.09	-.08	-.10	-.10	-.10	-.11	-.10	-.10

Table B4. Residual Horizontal Targeting Errors per Screen Location (θ, degrees)

Height	-70º	-50º	-30º	-10º	10º	30º	50º	70º
2.0 m	-0.3	-0.1	-0.1	-0.2	-0.1	0.3	-0.3	0.2
1.5 m	-0.3	-0.3	0.1	-0.1	-0.1	0.4	-0.1	0.4
1.0 m	0.3	-0.3	0.0	0.0	-0.1	0.5	-0.1	0.1
0.5 m	-0.1	-0.0	0.1	0.1	-0.2	0.1	-0.6	-0.1

Table B5. Residual Vertical Targeting Errors per Screen Location (y, meters)

Height	-70º	-50º	-30º	-10º	10º	30º	50º	70º
2.0 m	.00	.00	.00	.00	.00	.01	.00	-.01
1.5 m	.01	.01	.01	.01	.01	.01	.00	.00
1.0 m	.00	.01	-.01	.00	.00	.01	-.01	.00
0.5 m	.01	.01	-.01	.01	.01	.02	.00	-.01

Appendix C – Lua Scripting Code for Targeting Correction

```
-- get inputs
mfx = inputs.get("MFX")
mfy = inputs.get("MFY")
mfz = inputs.get("MFZ")
mbx = inputs.get("MBX")
mby = inputs.get("MBY")
mbz = inputs.get("MBZ")
gdt = inputs.get("GDT")
gdy = inputs.get("GDY")
cor = inputs.get("COR")
dbg = inputs.get("DBG")
dbt = inputs.get("DBT")
dby = inputs.get("DBY")
dbz = inputs.get("DBZ")

-- initializations
if action() == "Start" then
    -- initial values for dt_null, dy_null and rr need to be accurate
    dt_null = 0.8   -- screen center null theta (degrees)
    dy_null = -0.13 -- screen center null y (meters)
    rr = 2.5        -- projection screen radius (meters)

    -- these initial values are arbitrary and will be overwritten
    sct = 0        -- screen intercept theta
    scx = 0        -- screen intercept x (meters)
    scy = 1.25     -- screen intercept y (meters)
    scz = -rr      -- screen intercept z (meters)
    mlt = 1        -- marker vector to screen intercept multiplier
    delta_t = 0    -- screen correction theta (degrees)
    delta_y = 0    -- screen correction y (meters)
    delta_mx = 0   -- marker correction x (millimeters)
    delta_my = 0   -- marker correction y (millimeters)
    delta_mz = 0   -- marker correction z (millimeters)
    bCorrect = true

    --  base arrays for correction grids
    dtf = {}   dyf = {}   dtb = {}   dyb = {}   dt = {}   dy = {}

    --  screen theta corrections array
    dt[1] ={}   dt[2] ={}   dt[3] ={}   dt[4] ={}   dt[5] ={}

    --  screen y correction array
    dy[1] ={}   dy[2] ={}   dy[3] ={}   dy[4] ={}   dy[5] ={}

    --           screen t correction array @ bmz = -0.40m (degrees)
    --         -80   -60   -40   -20    0    +20   +30   +40   +60   +80
    dtf[1] ={ -0.8, -0.8, -0.8, -0.5,  0.0, -0.1,  0.3, -0.6, -0.5, -0.1}   -- 2.25
    dtf[2] ={ -0.4,  0.0, -0.5, -0.5,  0.0, -0.1,  0.2, -0.6, -0.7,  0.0}   -- 1.75
    dtf[3] ={ -0.4,  0.1, -0.2, -0.4,  0.1,  0.1,  0.4, -0.2, -0.6,  0.0}   -- 1.25
    dtf[4] ={ -1.0, -0.1, -0.6, -0.5,  0.1,  0.3,  0.6,  0.0,  0.1,  0.7}   -- 0.75
    dtf[5] ={ -2.1, -1.1, -0.8, -0.9, -0.3,  0.3,  0.4,  0.3,  0.6,  2.1}   -- 0.25

    --            screen y correction array @ bmz = -0.40m (meters)
    --         -80   -60   -40   -20    0    +20   +30   +40   +60   +80
    dyf[1] ={ -.04, -.05, -.05, -.05, -.04, -.03, -.03, -.04, -.04, -.03}   -- 2.25
    dyf[2] ={ -.01, -.04, -.02, -.02, -.01,  .00,  .00, -.01,  .00, -.01}   -- 1.75
    dyf[3] ={  .00, -.02, -.02,  .00,  .00,  .01,  .01,  .00,  .00, -.01}   -- 1.25
    dyf[4] ={ -.03, -.04, -.04, -.05, -.05, -.04, -.03, -.05, -.01, -.05}   -- 0.75
    dyf[5] ={ -.07, -.09, -.08, -.10, -.10, -.10, -.10, -.11, -.10, -.10}   -- 0.25
```

```
60
61      --           screen t correction array @ bmz = 0.10m (degrees)
62      --         -80   -60   -40   -20    0    +20   +30   +40   +60   +80
63      dtb[1] ={ -1.0, -1.3, -1.2, -0.8, -0.1, -0.4, -0.1, -0.6, -0.4,  0.1}   -- 2.25
64      dtb[2] ={ -1.1, -0.9, -1.2, -0.8,  0.1,  0.0,  0.1, -0.2, -0.3,  0.2}   -- 1.75
65      dtb[3] ={ -0.8, -0.9, -0.8, -0.7,  0.1,  0.0,  0.3, -0.3, -0.1,  0.3}   -- 1.25
66      dtb[4] ={ -0.8, -0.5, -0.5, -0.6,  0.1, -0.2,  0.2, -0.5, -0.1,  0.3}   -- 0.75
67      dtb[5] ={ -0.8, -0.5, -0.4, -0.5, -0.1, -0.4, -0.4, -0.7, -0.5,  0.2}   -- 0.25
68
69      --            screen y correction array @ bmz = 0.10m (meters)
70      --         -80   -60   -40   -20    0    +20   +30   +40   +60   +80
71      dyb[1] ={ -.04, -.04, -.04, -.04, -.03, -.03, -.02, -.04, -.03, -.03}   -- 2.25
72      dyb[2] ={ -.02, -.02, -.02, -.01, -.01,  .00,  .00,  .00,  .00,  .00}   -- 1.75
73      dyb[3] ={  .00, -.01, -.01, -.02,  .00,  .00,  .01,  .00,  .00, -.02}   -- 1.25
74      dyb[4] ={ -.03, -.03, -.05, -.04, -.03, -.04, -.02, -.03, -.04, -.06}   -- 0.75
75      dyb[5] ={ -.06, -.06, -.06, -.04, -.05, -.05, -.04, -.04, -.07, -.10}   -- 0.25
76  end
77
78  -- define 2D interpolation function
79  function interpolate(t, y, t1, t2, y1, y2, dt1y1, dt2y1, dt1y2, dt2y2)
80      if math.abs(t2 - t1) < 0.001 then
81          dy1 = dt1y1
82          dy2 = dt1y2
83      else
84          dy1 = dt1y1*(t2-t)/(t2-t1) + dt2y1*(t-t1)/(t2-t1)
85          dy2 = dt1y2*(t2-t)/(t2-t1) + dt2y2*(t-t1)/(t2-t1)
86      end
87
88      if math.abs(y2 - y1) < 0.001 then
89          dty = dy1
90      else
91          dty = dy1*(y2-y)/(y2-y1) + dy2*(y-y1)/(y2-y1)
92      end
93
94      return dty
95  end
96
97  -- pointer vector lengths and angles
98  vx = mfx - mbx
99  vy = mfy - mby
100  vz = mfz - mbz
101  vxz  = (vx^2 + vz^2)^0.5
102  vxyz = (vx^2 + vy^2 + vz^2)^0.5    -- actual length
103  vt = math.deg(math.atan2(vx,-vz))  -- horizontal angle
104  vp = math.deg(math.asin(vy/vxyz))  -- vertical angle
105
106  -- skip if pointer is near vertical or pointing backwards
107  if math.abs(vp) > 60 or math.abs(vt) > 110 then
108      scx = 0.0
109      scy = 0.0
110      scz = -rr
111      sct = 0.0
112      scp = 0.0
113  else
114      -- screen intersection x,y,z
115      aa = vx^2 + vz^2
116      bb = 2 * (vx * mbx + vz * mbz)
117      cc = mbx^2 + mbz^2 - rr^2
118      ds = bb^2 - 4 * aa * cc
119
120      if ds > 0 then
121          mlt = (ds^0.5 - bb) / (2 * aa)
```

```
122         scx = mbx + vx * mlt
123         scy = mby + vy * mlt + dy_null
124         scz = mbz + vz * mlt
125     end
126 
127     -- screen intersection theta
128     sct = math.deg(math.atan2(scx,-scz))
129 end
130 
131 -- debug with slider inputs
132 if dbg == 1 then
133     dt_null = 0.0
134     dy_null = 0.0
135     if dbt >  100 then dbt =  100 end
136     if dbt < -100 then dbt = -100 end
137     if dby >    3 then dby =    3 end
138     if dby <   -1 then dby =   -1 end
139     sct = dbt
140     scy = dby
141     mbz = dbz
142 end
143 
144 -- create working array by interpolating forward and aft arrays per mbz
145 if mbz < -.45 then mbz = -.45 end
146 if mbz >  .20 then mbz =  .20 end
147 
148 for i=1,5,1 do
149     for j=1,9,1 do
150         dt[i][j] = dtb[i][j] + (dtf[i][j]-dtb[i][j]) * (0.25-mbz)/(0.25+0.30)
151         dy[i][j] = dyb[i][j] + (dyf[i][j]-dyb[i][j]) * (0.25-mbz)/(0.25+0.30)
152     end
153 end
154 
155 -- assign interpolation parameters based on screen position
156 if sct > 60 then
157     t1 = 60
158     t2 = 80
159     inb1 = 9
160     inb2 = 10
161 elseif sct > 40 then
162     t1 = 40
163     t2 = 60
164     inb1 = 8
165     inb2 = 9
166 elseif sct > 30 then
167     t1 = 30
168     t2 = 40
169     inb1 = 7
170     inb2 = 8
171 elseif sct > 20 then
172     t1 = 20
173     t2 = 30
174     inb1 = 6
175     inb2 = 7
176 elseif sct > 0 then
177     t1 = 0
178     t2 = 20
179     inb1 = 5
180     inb2 = 6
181 elseif sct > -20 then
182     t1 = -20
183     t2 = 0
```

```
184      inb1 = 4
185      inb2 = 5
186  elseif sct > -40 then
187      t1 = -40
188      t2 = -20
189      inb1 = 3
190      inb2 = 4
191  elseif sct > -60 then
192      t1 = -60
193      t2 = -40
194      inb1 = 2
195      inb2 = 3
196  else
197      t1 = -80
198      t2 = -60
199      inb1 = 1
200      inb2 = 2
201  end
202
203  if scy > 1.75 then
204      y1 = 2.25
205      y2 = 1.75
206      ina1 = 1
207      ina2 = 2
208  elseif scy > 1.25 then
209      y1 = 1.75
210      y2 = 1.25
211      ina1 = 2
212      ina2 = 3
213  elseif scy > 0.75 then
214      y1 = 1.25
215      y2 = 0.75
216      ina1 = 3
217      ina2 = 4
218  else
219      y1 = 0.75
220      y2 = 0.25
221      ina1 = 4
222      ina2 = 5
223  end
224
225  -- call interpolation function for delta theta and delta phi
226  offset_t = interpolate(sct, scy, t1, t2, y1, y2, dt[ina1][inb1],
227                  dt[ina1][inb2], dt[ina2][inb1], dt[ina2][inb2])
228
229  offset_y = interpolate(sct, scy, t1, t2, y1, y2, dy[ina1][inb1],
230                  dy[ina1][inb2], dy[ina2][inb1], dy[ina2][inb2])
231
232  if cor == 0 or dbg == 1 then
233      -- apply null corrections plus array values
234      delta_t = dt_null + offset_t
235      delta_y = dy_null + offset_y
236      bCorrect = true
237  elseif cor == 1 then
238      -- apply null corrections plus game controller inputs
239      delta_t = dt_null + gdt
240      delta_y = dy_null - gdy
241      bCorrect = true
242  elseif cor == 2 then
243      -- apply null corrections plus array values plus game controller inputs
244      delta_t = dt_null + offset_t + gdt
245      delta_y = dy_null + offset_y - gdy
```

```
246      bCorrect = true
247  else
248      -- do not apply any corrections
249      bCorrect = false
250  end
251
252
253  -- convert delta_t to delta_mx and delta_mz
254  dsp_xz = (2*(vxz)^2 * (1 - math.cos(math.rad(delta_t))))^0.5
255  if delta_t < 0 then dsp_xz = -dsp_xz end
256  delta_mx = dsp_xz * math.cos(math.rad(vt))
257  delta_mz = dsp_xz * math.sin(math.rad(vt))
258
259  -- convert delta_y to delta_my
260  delta_my = delta_y / mlt
261
262  -- apply the corrections to forward marker position
263  if bCorrect then
264      mfx = mfx + delta_mx
265      mfy = mfy + delta_my
266      mfz = mfz + delta_mz
267  end
268
269
270  -- set outputs
271  outputs.set("MkrFX", mfx)
272  outputs.set("MkrFY", mfy)
273  outputs.set("MkrFZ", mfz)
274  outputs.set("MkrBX", mbx)
275  outputs.set("MkrBY", mby)
276  outputs.set("MkrBZ", mbz)
277  outputs.set("SCT", sct)
278  outputs.set("SCX", scx)
279  outputs.set("SCY", scy)
280  outputs.set("SCZ", scz)
281
282  if cor == 0 or cor == 1 then
283      outputs.set("DeltaT", delta_t - dt_null)
284      outputs.set("DeltaY", delta_y - dy_null)
285  elseif cor == 2 then
286      outputs.set("DeltaT", gdt)
287      outputs.set("DeltaY", -gdy)
288  else
289      outputs.set("DeltaT", 0)
290      outputs.set("DeltaY", 0)
291  end
292
293  outputs.set("DeltaMX", delta_mx)
294  outputs.set("DeltaMY", delta_my)
295  outputs.set("DeltaMZ", delta_mz)
```

REPORT DOCUMENTATION PAGE

1. REPORT DATE (DD MM YY)
24 10 13

2. REPORT TYPE
Technical Report

3. DATES COVERED (from – to)
Apr 2013 – Aug 2013

4. TITLE
Improvements in Targeting Precision by Applying Pointing Corrections to a Computer Assisted Rehabilitation Environment (CAREN)

5a. Contract Number:
5b. Grant Number:
5c. Program Element Number:
5d. Project Number:
5e. Task Number:
5f. Work Unit Number: 61030

6. AUTHORS
Johnson, Delwin O.; Sessoms, Pinata H.; Reini, Seth A. LT

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Commanding Officer
Naval Health Research Center
140 Sylvester Rd
San Diego, CA 92106-3521

8. PERFORMING ORGANIZATION REPORT NUMBER
13-58

9. SPONSORING/MONITORING AGENCY NAMES(S) AND ADDRESS(ES)
Commanding Officer
Naval Medical Research Center
503 Robert Grant Ave
Silver Spring, MD 20910-7500

Chief, Bureau of Medicine and Surgery
(MED 00), Navy Dept
2300 E Street NW
Washington, DC 20372-5300

10. SPONSOR/MONITOR'S ACRONYM(S)
BUMED/NMRC

11. SPONSOR/MONITOR'S REPORT NUMBER(s)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

The Computer Assisted Rehabilitation Environment (CAREN) at the Naval Health Research Center (NHRC) is an interactive virtual environment that has the ability to task subjects both cognitively and physically. The CAREN D-Flow software (Motek Medical BV, Amsterdam, The Netherlands) that allows the operator to create and control the scenarios has different modules from which to manipulate and monitor the hardware components, activate events, or record information. The Pointer3D module targets objects in a CAREN scene by extrapolating the position of two markers attached to a simulated weapon or other pointing device. The typical result has been that the projected target does not align well with the pointing markers, thus limiting the possibility of using the CAREN system for precision targeting such as in the case of a shooting simulator. The purpose of this paper is to apply effective targeting corrections in a manner that is portable between D-Flow applications and also maintainable for changing situations.

The initial implementation of pointing corrections as described in this document produced favorable results overall for the NHRC CAREN. Targeting correction technique described in this paper offers credence that the CAREN could be used as an effective shooting simulator or at least incorporate realistic targeting within the CAREN software applications.

15. SUBJECT TERMS
targeting, pointing corrections, CAREN, Naval Health Research Center, NHRC, Motek Medical, shooting, simulated weapons, virtual environments

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	18a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Commanding Officer
UNCL	UNCL	UNCL	UNCL	17	18b. TELEPHONE NUMBER (INCLUDING AREA CODE) COMM/DSN: (619) 553-8429

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39-18